IMPLEMENTATION OF SDN FIREWALL IN PYTHON USING MININET

Report submitted to the SASTRA Deemed to be University
as the requirement for the course

CSE302: COMPUTER NETWORKS

Submitted by

YANAMADALA UJJWALA
124157077,B.Tech - CSE CSBC

DECEMBER 2022

SCHOOL OF COMPUTING

THANJAVUR, TAMIL NADU, INDIA – 613 401

SCHOOL OF COMPUTING

THANJAVUR – 613 401

Bonafide Certificate

This is to certify that the report titled "**IMPLEMENTATION OF SDN FIREWALL IN PYTHON USING MININET**" submitted as a requirement for the course, **CSE302: COMPUTER NETWORKS** for B.Tech. is a bonafide record of the work done by **Shri. Yanamadala Ujjwala (Reg. No.124157077,CSE CSBC**) during the academic year 2022-23, in the School of Computing

Project Based Work *Viva voc*e held on

Examiner 1 Examiner 2

Table of contents:

List of Figures

iii
Abbreviations

OVS - Open V Switch

IP - Internet Protocol

TCP-Transmission Control Protocol

ICMP - Internet Control Message ProtocoL

SDN -Software Defined Network

LAN - Local Area Network

iv
Abstract

Software Defined Network is a networking architecture approach. It enables the control and management of networks using software applications. Through Software Defined Networking behavior of the entire network and its devices are programmed in centrally controlled manner through software applications using open APIs.Traditional network refers to the old conventional way of networking which uses fixed and dedicated hardware devices such as routers and switches to control network traffic. Inability to scale and network security and Performance are the major concerns nowadays in the current growing business situation so that SDN is taking control of traditional networks.

A firewall is a system that secures incoming network packets, which come from various sources, as well as outgoing network packets. It can monitor and control the flow of data which comes into the network from different sources, and works on the basis of predefined rules.They can be categorized as either hardware or software firewalls. Network firewalls are software programs running on different hardware appliances in the network.Software based firewalls provide a layer of software on a host, which controls network traffic in and out of that particular machine.The firewall policy is also centrally defined and enforced at the controller.

SDN could be used to replace expensive Layer 4-7 firewalls, load-balancers, and IPS/IDS with cost-effective high-performance switches with a logically centralized controller. In this project, I have implemented Layer 2, 3 & 4 firewall by proactive policies and managed to block several applications like specific link, host-to-host connectivity and destination process by making use of POX controller and OVS switch in Mininet.

KEYWORDS: Firewall, Software Defined Network, Traditional Network, POX controller,OVS switch

V
1.INTRODUCTION

SDN is a new paradigm of computer network systems in which the network control plane is separated from the data plane and assigned to a dedicated software program called controller running at the control plane . First, SDN separates the control plane which decides how to handle the traffic, from the data plane underlying devices which forward traffic based on decisions that the control plane makes . Secondly with that separation of the control and data planes, network underlying devices become simple dump forwarding devices and the control logic is implemented in a logically centralized controller acting as network operating system NOS that creates a consistent and up-to-date network view. Thirdly, with the reinforcement of empowering control plane, SDN makes it possible for a single software program to control multiple infrastructure layer devices simultaneously. Decoupling control plane and data plane brings an advantage of an abstraction model which makes it easier for both control and data planes to develop separately as well as the ability to easily manage and program distributed underlying network devices .SDN shows signs of significantly facilitating network management and empowers novelty, innovation and advancement

OpenFlow is the first and the present industrial standardized SDN protocol defined between the control layer and data layer of SDN architecture. Using the OpenFlow protocol, a controller and a switch can communicate with each other as well as the controller can manage the switch over the secure channel . Moreover, OpenFlow makes it easy for the controller to deal with all network elements in the infrastructure layer and manage them efficiently .OpenFlow defines the way in which software applications can program the flow table of different switches .

Since the Firewall is the first line of defense, deploying firewalls to protect the networks against threats is essential and indispensable. In addition, it is the most broadly security mechanism used in most networks of organizations and foundations . Typically, a firewall sits on the border between LAN and public network (Internet), filtering all traffic passing through it. It inspects all incoming and outgoing packets then allows or denies based on predefined rules r91.

The OpenFlow switch has one or more flow tables. Each flow table contains a set of flow entries to match packets in priority order. Each flow entry consists of header values (to which packets are matched), counters (to update for matching packet) and a set of actions (to apply to matching packets).

Upon receiving incoming packet, the OpenFlow switch starts handling it through matching comparison

against the flow table(s)'s entries . Usually, matching starts at the first flow table and may continue to

additional flow tables in the pipeline in case of missing flow entry found. As a rule, packets match against flow entries in priority order. If a matching entry is found, any actions associated with the specific flow entry are executed on that packet (e.g., the action might be to forward a packet out a specified port). If no match is found, the packet is simply forwarded to the controller over the secure channel. The controller in turn informs the switch how to do with packets which have no valid flow entries. Mainly because the controller is in charge of adding, updating and deleting flow entries in flow tables as well as determining how to process such packets in such a case.

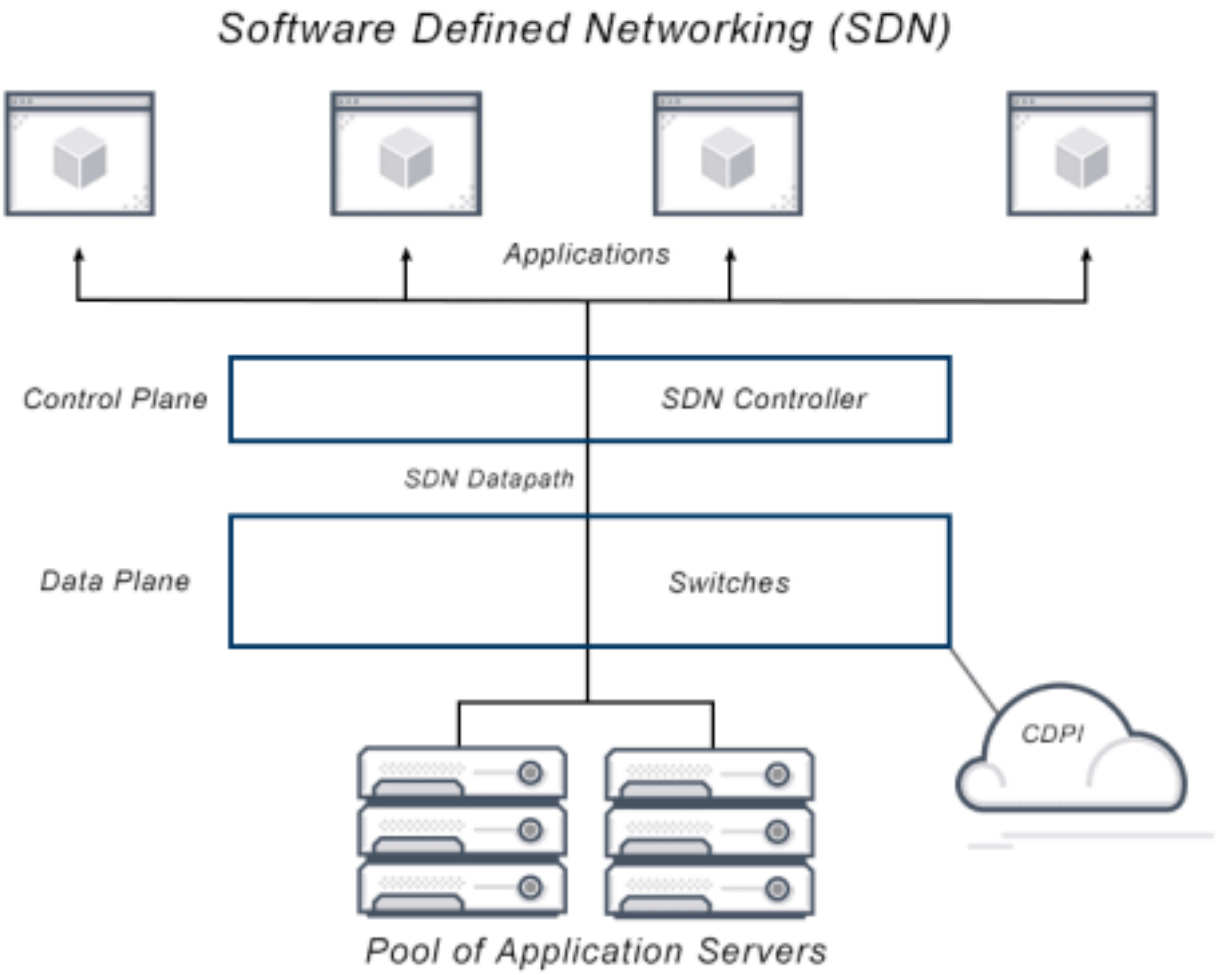

FIG:1.1 : SDN ARCHITECTURE

2.FEATURES IMPLEMENTED

In order to implement and test the functionality of our firewall, the following tools were used:

1. *VirtualBox* provides an environment for virtual networks to be formed. It is used to install Mininet VM.
2. *Mininet* is a network emulator to create realistic virtual SDN network topology . 3. POX Controller is an open source Python-based SDN controller which supports OpenFlow.

4. *Wireshark* provides testing and performance analysis.

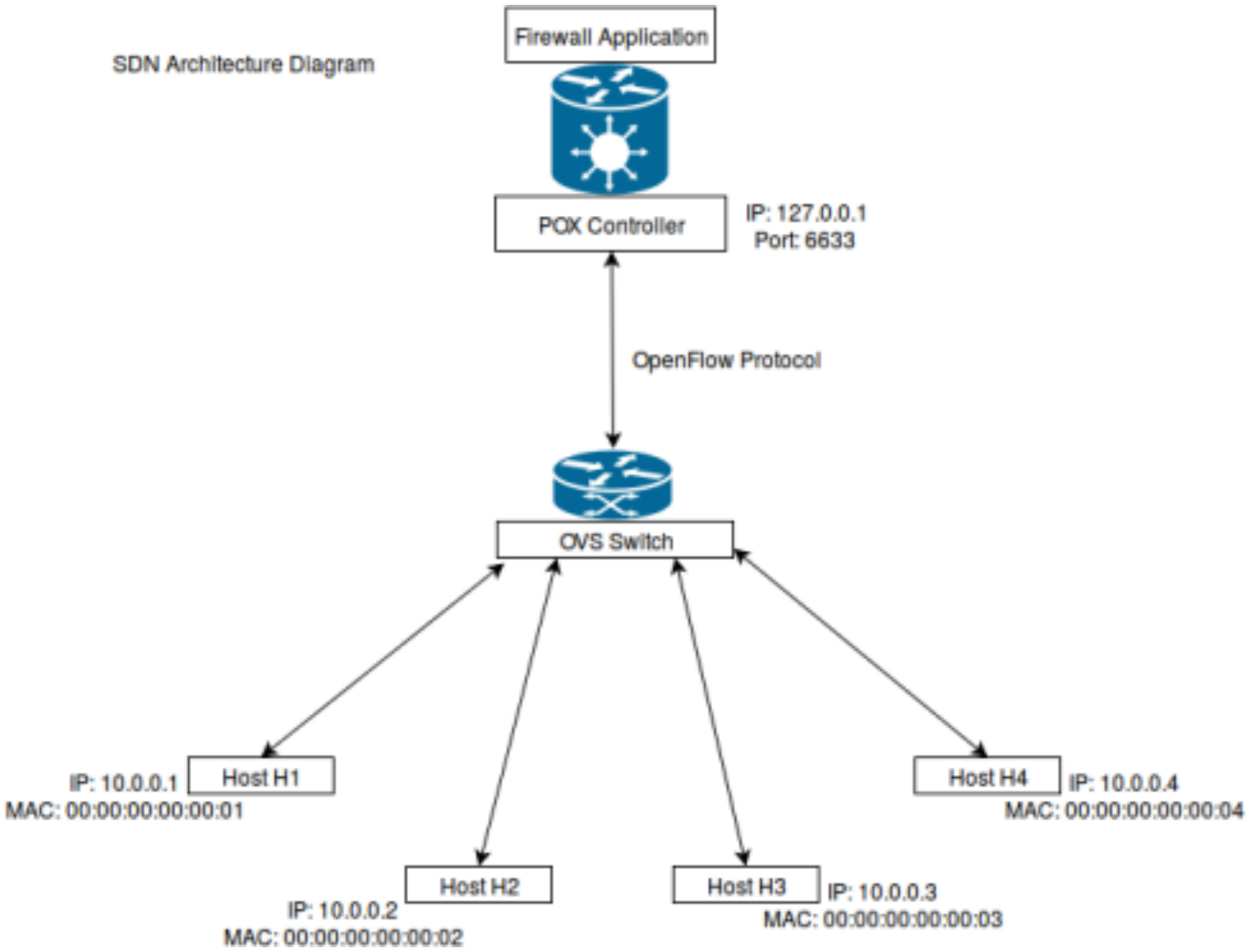

FIG:2.1:Network Model using SDN controller

The experiment was conducted using VirtualBox Which is used for virtualization. We set up a VM, running Mininet on the top of an Ubuntu OS. For creating network topology, a Mininet emulator was used.

Installing:

There are two scripts namely, mininetScript.py and poxController_firewall.py which will run the mininet based ovsSwitch and POX Controller respectively.

- Running the POX Controller

- Go to the pox directory installed in your setup and execute " ./pox.py log.level –INFO poxController_firewall"
- Running the Mininet and OVS-Switch
- Execute the python script " python mininetScript.py"

Firewall Rules installed in the controller:

- Layer - 2 Link connection blocked, Block all the traffic from Host-h2
- AddRule(dataPath_id,00:00:00:00:00:02,NULL,NULL,NULL)
- Layer - 3 Host-to-Host connectivity blocked, Block all the traffic from Host-h1 to Host-h4
- AddRule(dataPath_id,NULL,10.0.0.1,10.0.0.4,NULL)
- Layer - 4 Destination Process blocked, Block all the traffic destined to Process h3 at port 80
- AddRule(dataPath_id,NULL,NULL,10.0.0.3,80)

Running the Tests

- Layer-2 Link connection blocked
- H2 ping h4
- Layer -3 Host - to-Host Connectivity blocked
- Layer-4 Destination Process blocked
- Run two IPerf servers on Host-h
- h3 iperf -s -p 80&
- h3 iperf -s -p 80&
- Connect to both the servers
- h1 iperf -c h3 -p 80 -t 2 -i 1
- h1 iperf -c h3 -p 22 -t 2 -i 1

Two python scripts were created: the first python script " mininetScript.py" for our experimentation setup to create custom topology and the second one is "poxController_firewall.py". The " mininetScript.py" created topology by setting up one remote controller, one switch and four hosts connected to it as shown in Figure 2 Host4 having IP address 10.0.0.4 acts as a web server while the three other hosts: host1, host2 and host3 having IP address 10.0.0.1, 10.0.0.2 and 10.0.0.3 respectively act as clients. The Ethernet address of each host corresponds to its IP address for simplicity, meaning host1 has 00:00:00:00:00:01, host2 has 00:00:00:00:00:02, and so on. To run our " mininetScript.py", new terminal is opened and this command " python mininetScript.py" is typed to bring up one OpenFlow switch with the four hosts connected to it and configure it to connect to a remote controller POX.

The second python script is "poxController_firewall.py", it is our firewall module which first imports csv files containing the IP and MAC policies to be implemented. Another new terminal is opened and this command " ./pox.py log.level –INFO poxController_firewall" is typed to run the POX controller with our firewall application. Once POX controller is up and listening on port 6633 to which OpenFlow switch with Ethernet address [00-00-00-00-00-07] is connected, the firewall module "poxController_firewall.py" will be running on the top of POX and MAC and IP (TCP/UDP) policies are parsed and normally added.

There are three kinds of policies have been added:

1. MAC rule: it represents the Layer2 firewall which catches Ethernet header to check if it belongs to host1 (00:00:00: 00:00:01), it should be blocked in both directions.
2. IP rule: it represents the layer3 firewall which detects IP packet to check if it matches the rule, then it should not be allowed to forward; and here we specified blocking IP Host-to-Host

connectivity.

3. TCP and UDP rule: our firewall works here to catch Layer4 traffic to perform port security. Once it catches TCP segment or UDP datagram, the rules are executed accordingly.

3.RELATED WORK

The main responsibility of our firewall is to filter the packets based on their parsed headers then to permit or deny according to rules specified. Hence, it needs to run together with a switching module which is responsible to enforce the OpenFlow switch to act as a type of 12_learning switch. Yet running

these two modules exactly at the same time will cause OpenFlow error messages because both modules will try to access the same buffer ID . Furthermore, it is possible that different rules, flow table entries, might be installed to the OpenFlow switch by the two modules which might result in policy conflicts.

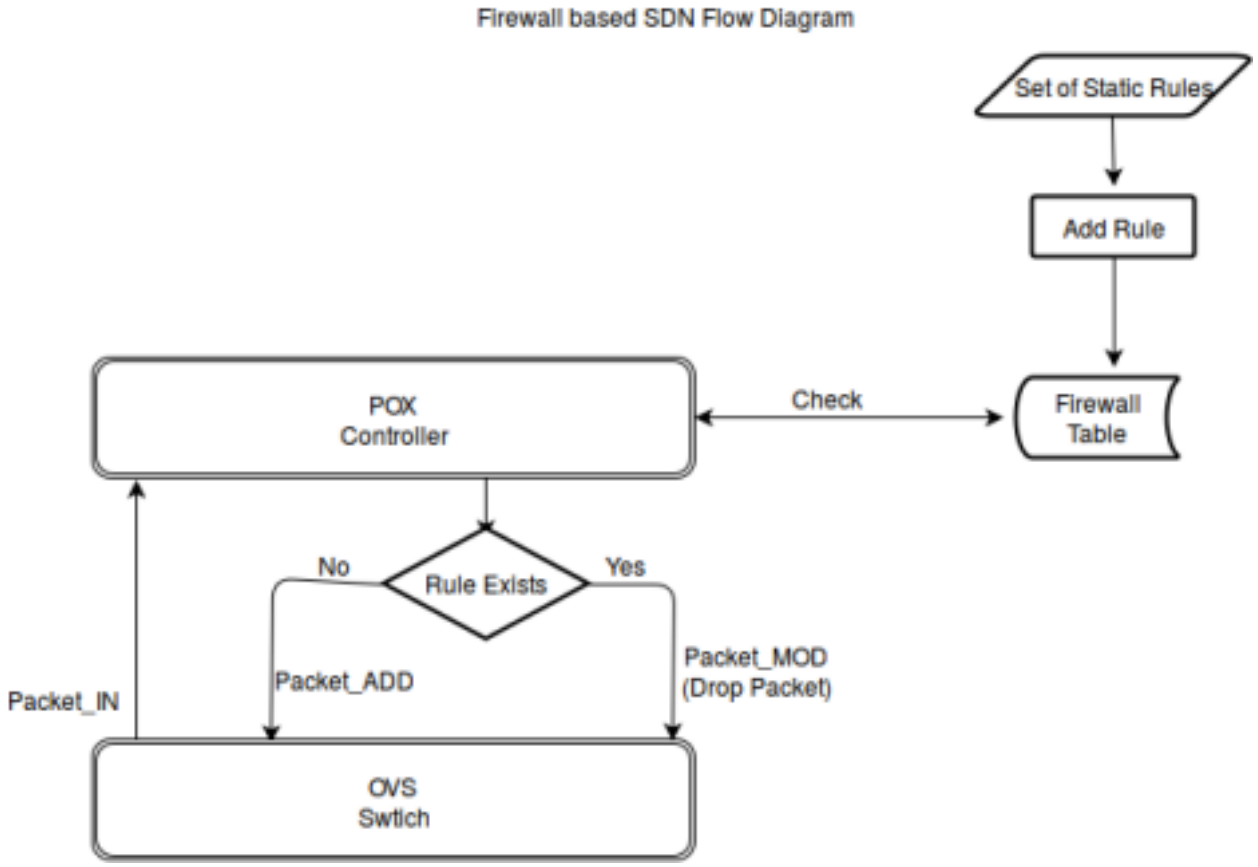

FIG:3.1:Firewall based SDN flow diagram

To solve the conflict issue, there are two methods: The first method is creating one class called “Master” which includes both the firewall module and the switch module. The rule of the “Master” class is organizing the order of calling the module's functions. The second method is keeping both modules separated but the switch module should be modified to listen to events fired by our firewall module in place of events triggered by OpenFlow protocol . In our work, the second mode is implemented. The 12_learning switch is chosen to run along with our firewall module and modified to listen to its triggered events. Thus, both modules will work together in harmony.

It is clear from the details of Flowcharts in Figure 3.1 that the firewall application starts parsing the firewall policies then listens to catch a new event -packet- sent from a switch. Upon receiving the packet, the firewall application parses the packet to take out the Ethernet header and learn or update the table that contains switch ports associated with MAC addresses, and then check whether it is an IP packet or ARP request. If it is an IP packet, it checks whether it is ICMP in case of "ping" or TCP segment or UDP datagram and blocks or allows according to the specified rules. If it is an ARP request, it will send a request to get ARP reply then stores flow table entries on switches to speed up packet-processing for further packets and to reduce bandwidth consumption of controller-switch link.

Since this firewall is stateless, it does not keep any state of information to track down ongoing connections. The learning switch algorithm is modified to perform the action taken by the firewall.

4.SOURCE CODE

4.1.Configuring POX Controller:

```
 from pox.core import core

import pox.openflow.libopenflow_01 as of

from pox.lib.util import dpid_to_str

from pox.lib.util import str_to_bool

from pox.lib.packet.arp import arp

from pox.lib.packet.ipv4 import ipv4

from pox.lib.packet.ipv6 import ipv6

from pox.lib.addresses import IPAddr, EthAddr

import time

log = core.getLogger()

_flood_delay = 0
```

```
class LearningSwitch (object):
def __init__ (self, connection, transparent):
    # Switch we'll be adding L2 learning switch capabilities to
    self.connection = connection
```



```
    self.transparent = transparent

    # Our table
    self.macToPort = {}

    # Our firewall table in the form of Dictionary
    self.firewall = {}

    # Add a Couple of Rules Static entries
    # Two type of rules: (srcip,dstip) or (dstip,dstport)
    self.AddRule(dpid_to_str(connection.dpid), EthAddr('00:00:00:00:00:02'), 0, 0, 0)
    self.AddRule(dpid_to_str(connection.dpid), 0, IPAddr('10.0.0.1'), IPAddr('10.0.0.4'),0)
    self.AddRule(dpid_to_str(connection.dpid), 0, 0, IPAddr('10.0.0.3'), 80)

    # We want to hear PacketIn messages, so we listen
    # to the connection
    connection.addListeners(self)

    # We just use this to know when to log a helpful message
```

```
  self.hold_down_expired = _flood_delay == 0

# function that allows adding firewall rules into the firewall table
def AddRule (self, dpidstr, macstr, srcipstr, dstipstr, dstport,value=True):
    if srcipstr == 0 and dstipstr == 0:
      self.firewall[(dpidstr,macstr)] = True
```



```
      log.debug("Adding L2-firewall rule of Src(%s) in %s", macstr, dpidstr)
    elif dstport == 0:
      self.firewall[(dpidstr,srcipstr,dstipstr)] = True
      log.debug("Adding L3-firewall rule of %s -> %s in %s", srcipstr, dstipstr,
    dpidstr) elif srcipstr == 0:
      self.firewall[(dpidstr,dstipstr,dstport)] = True
      log.debug("Adding L4-firewall rule of Dst(%s,%s) in %s", dstipstr, dstport,
    dpidstr) else:
      self.firewall[(dpidstr,srcipstr,dstipstr,dstport)] = True
      log.debug("Adding firewall rule of %s -> %s,%s in %s", srcipstr, dstipstr, dstport, dpidstr)

# function that allows deleting firewall rules from the firewall table
def DeleteRule (self, dpidstr, macstr, srcipstr, dstipstr, dstport):
   try:
     if srcipstr == 0 and dstipstr == 0:
       del self.firewall[(dpidstr,macstr)]
       log.debug("Deleting L2-firewall rule of Src(%s) in %s", macstr, dpidstr)
     elif dstport == 0:
```

```
      del self.firewall[(dpidstr,srcipstr,dstipstr)]
      log.debug("Deleting L3-firewall rule of %s -> %s in %s", srcipstr, dstipstr,
    dpidstr) elif srcipstr == 0:
      del self.firewall[(dpidstr,dstipstr,dstport)]
      log.debug("Deleting L4-firewall rule of Dst(%s,%s) in %s", dstipstr, dstport,
    dpidstr) else:
      del self.firewall[(dpidstr,srcipstr,dstipstr,dstport)]
```



```
      log.debug("Deleting firewall rule of %s -> %s,%s in %s", srcipstr, dstipstr, dstport,
   dpidstr) except KeyError:
     log.error("Cannot find Rule %s(%s) -> %s,%s in %s", srcipstr, macstr, dstipstr, dstport, dpidstr)

# check if packet is compliant to rules before proceeding
def CheckRule (self, dpidstr, macstr, srcipstr, dstipstr, dstport):
  # Source Link blocked
  try:
    entry = self.firewall[(dpidstr, macstr)]
    log.info("L2-Rule Src(%s) found in %s: DROP", macstr, dpidstr)
    return entry
  except KeyError:
    log.debug("Rule Src(%s) NOT found in %s: L2-Rule NOT found", macstr, dpidstr)

  # Host to Host blocked
  try:
    entry = self.firewall[(dpidstr, srcipstr, dstipstr)]
```

```
    log.info("L3-Rule (%s x->x %s) found in %s: DROP", srcipstr, dstipstr,
    dpidstr) return entry
  except KeyError:
    log.debug("Rule (%s -> %s) NOT found in %s: L3-Rule NOT found", srcipstr, dstipstr, dpidstr)

 # Destination Process blocked
 try:
    entry = self.firewall[(dpidstr, dstipstr, dstport)]
```



```
    log.info("L4-Rule Dst(%s,%s)) found in %s: DROP", dstipstr, dstport,
    dpidstr) return entry
  except KeyError:
    log.debug("Rule Dst(%s,%s) NOT found in %s: L4-Rule NOT found", dstipstr, dstport,
    dpidstr) return False

def _handle_PacketIn (self, event):
  """
  Handle packet in messages from the switch to implement above algorithm.
  """
  packet = event.parsed
  inport = event.port

  def flood (message = None):
    """ Floods the packet """
    msg = of.ofp_packet_out()
```

```
if time.time() - self.connection.connect_time >= _flood_delay:
  # Only flood if we've been connected for a little while...

  if self.hold_down_expired is False:
    # Oh yes it is!
    self.hold_down_expired = True
    log.info("%s: Flood hold-down expired -- flooding", dpid_to_str(event.dpid))

  if message is not None: log.debug(message)
```



```
  msg.actions.append(of.ofp_action_output(port = of.OFPP_FLOOD))
 else:
  pass
  #log.info("Holding down flood for %s", dpid_to_str(event.dpid))
 msg.data = event.ofp
 msg.in_port = event.port
 self.connection.send(msg)

def drop (duration = None):
 """
 Drops this packet and optionally installs a flow to continue
 dropping similar ones for a while
 """
 if duration is not None:
  if not isinstance(duration, tuple):
```

```
    duration = (duration,duration)
   msg = of.ofp_flow_mod()
   msg.match = of.ofp_match.from_packet(packet)
   msg.idle_timeout = duration[0]
   msg.hard_timeout = duration[1]
   msg.buffer_id = event.ofp.buffer_id
   self.connection.send(msg)
  elif event.ofp.buffer_id is not None:
   msg = of.ofp_flow_mod() #creats a flow modification message
   msg.match = of.ofp_match.from_packet(event.parsed, event.port)
```



```
   msg.match.dl_dst = None
   msg.idle_timeout = 120
   msg.hard_timeout = 120
   msg.priority = 65535 #priority at which a rule will match, higher is better.
   msg.command = of.OFPFC_MODIFY
   msg.flags = of.OFPFF_CHECK_OVERLAP
   msg.data = event.ofp
   self.connection.send(msg)# send the message to the OpenFlow switch

self.macToPort[packet.src] = event.port # 1

# Get the DPID of the Switch Connection
dpidstr = dpid_to_str(event.connection.dpid)
#log.debug("Connection ID: %s" % dpidstr)
```

```
    if isinstance(packet.next, ipv4):
      log.debug("%i IP %s => %s", inport, packet.next.srcip,packet.next.dstip)
      segmant = packet.find('tcp')
      if segmant is not None:
       # Check the Firewall Rules in MAC, IPv4 and TCP Layer
           if self.CheckRule(dpidstr, packet.src, packet.next.srcip, packet.next.dstip, segmant.dstport) ==
True:
         drop()
         return
      else:
       # Check the Firewall Rules in MAC and IPv4 Layer
```



```
       if self.CheckRule(dpidstr, packet.src, packet.next.srcip, packet.next.dstip, 0) ==
        True: drop()
        return
    elif isinstance(packet.next, arp):
     # Check the Firewall Rules in MAC Layer
     if self.CheckRule(dpidstr, packet.src, 0, 0, 0) == True:
      drop()
      return
     a = packet.next
                                                    log.debug("%i ARP %s %s => %s", inport,
{arp.REQUEST:"request",arp.REPLY:"reply"}.get(a.opcode, 'op:%i' % (a.opcode,)), str(a.protosrc),
str(a.protodst))
    elif isinstance(packet.next, ipv6):
```

```
  # Do not handle ipv6 packets
  return

 if not self.transparent: # 2
  if packet.type == packet.LLDP_TYPE or packet.dst.isBridgeFiltered():
   drop() # 2a
   return

 if packet.dst.is_multicast:
  flood() # 3a
 else:
  if packet.dst not in self.macToPort: # 4
   flood("Port for %s unknown -- flooding" % (packet.dst,)) # 4a
```



```
  else:
   port = self.macToPort[packet.dst]
   if port == event.port: # 5
    # 5a
      log.warning("Same port for packet from %s -> %s on %s.%s. Drop." % (packet.src, packet.dst,
dpid_to_str(event.dpid), port))
    drop(10)
    return
   # 6
   log.debug("installing flow for %s.%i -> %s.%i" % (packet.src, event.port, packet.dst,
   port)) msg = of.ofp_flow_mod()
   msg.match = of.ofp_match.from_packet(packet, event.port)
```

```
        msg.idle_timeout = 10
        msg.hard_timeout = 30
        msg.actions.append(of.ofp_action_output(port = port))
        msg.data = event.ofp # 6a
        self.connection.send(msg)

class l2_learning (object):
  """
  Waits for OpenFlow switches to connect and makes them learning switches.
  """
  def __init__ (self, transparent):
    core.openflow.addListeners(self)
    self.transparent = transparent
```



```
  def _handle_ConnectionUp (self, event):
    log.debug("Connection %s" % (event.connection,))
    LearningSwitch(event.connection, self.transparent)

def launch (transparent=False, hold_down=_flood_delay):
  """
  Starts an L2 learning switch.
  """
  try:
    global _flood_delay
```

```
  _flood_delay = int(str(hold_down), 10)
  assert _flood_delay >= 0
 except:
  raise RuntimeError("Expected hold-down to be a number")

 core.registerNew(l2_learning, str_to_bool(transparent))
```

4.2.Network Topology:

```
from mininet.topo import Topo
from mininet.net import Mininet
from mininet.util import dumpNodeConnections
from mininet.log import setLogLevel
from mininet.node import RemoteController
```



```
from mininet.cli import CLI

class SingleSwitchTopo(Topo):
   "Single switch connected to n hosts."
   def build(self, n=2):
      switch = self.addSwitch('s1', dpid="00000000000007")
      # Python's range(N) generates 0..N-1
      for h in range(n):
            host = self.addHost('h%s' % (h + 1), ip='10.0.0.%s' % (h+1), mac='00:00:00:00:00:0%s' %
(h+1))
```

```
            self.addLink(host, switch)


def simpleTest():
    "Create and test a simple network"
    topo = SingleSwitchTopo(n=4)
    net = Mininet(topo, controller=None)
    " Remote POX Controller"
    c = RemoteController('c', '0.0.0.0', 6633)
    net.addController(c)
    net.start()
    print "Dumping host connections"
    dumpNodeConnections(net.hosts)
    CLI(net)
    net.stop()


if __name__ == '__main__':
```



```
    # Tell mininet to print useful information
    setLogLevel('info')
    simpleTest()
```

5.SNAPSHOTS

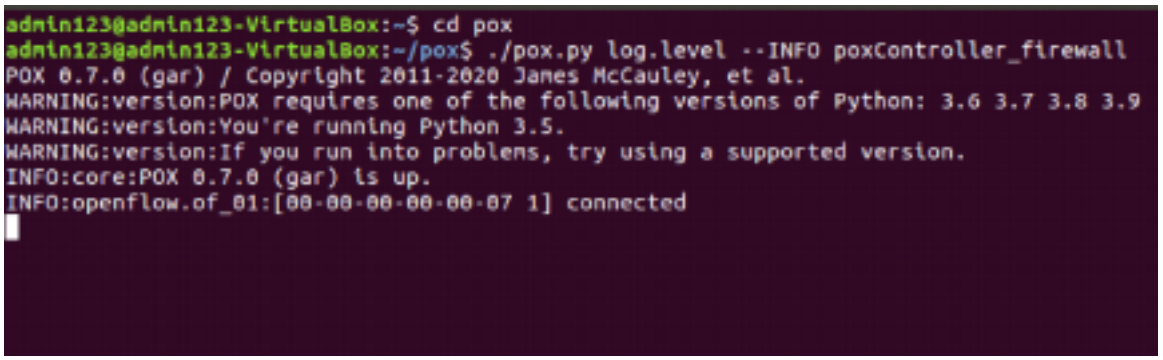

FIG:5.1: STARTING THE POX CONTROLLER

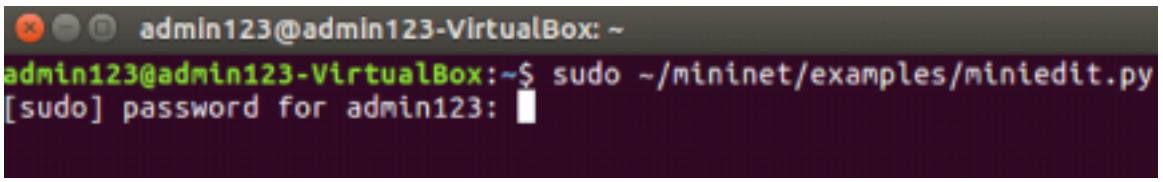

FIG:5.2: STARTING MINIEDIT GUI

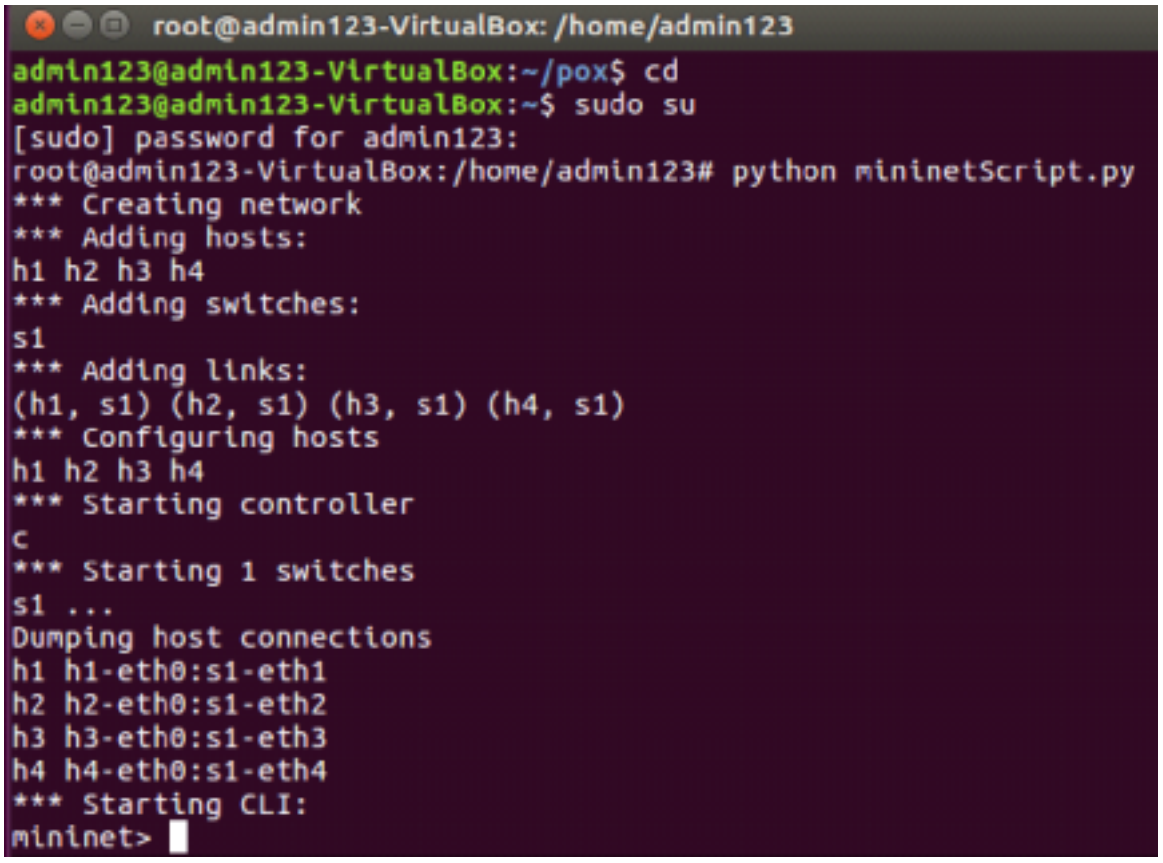

FIG:5.3: RUNNING NETWORK TOPOLOGY USING mininetScript IN PYTHON

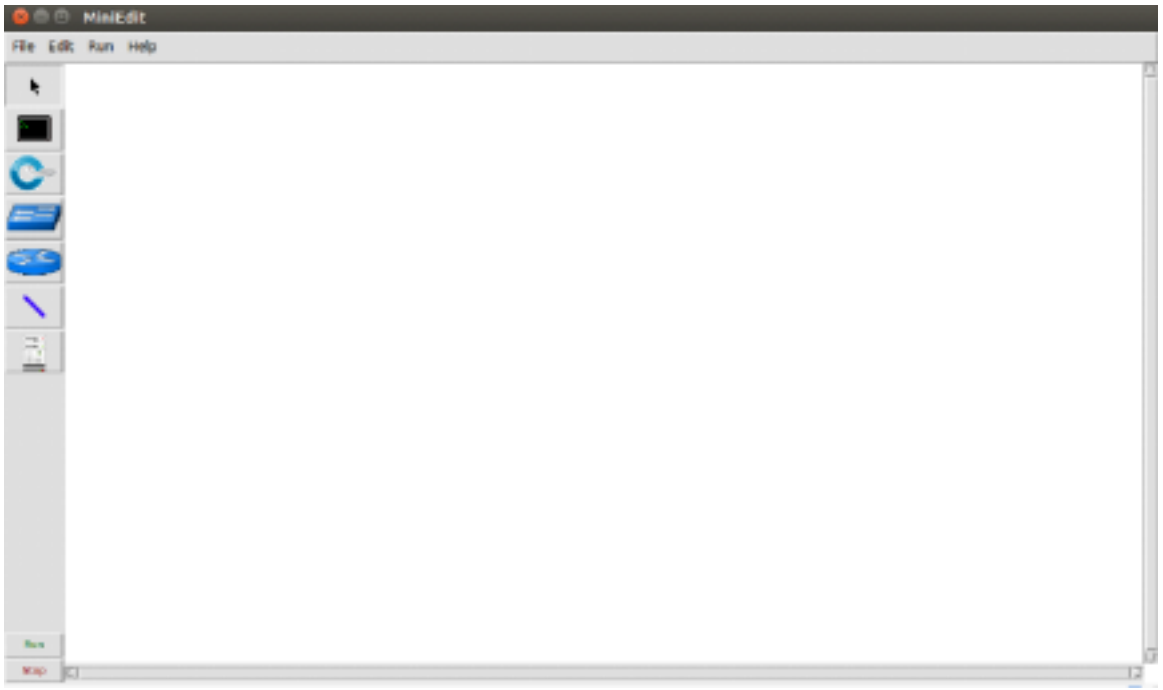

FIG:5.4: MINIEDIT GUI INTERFACE

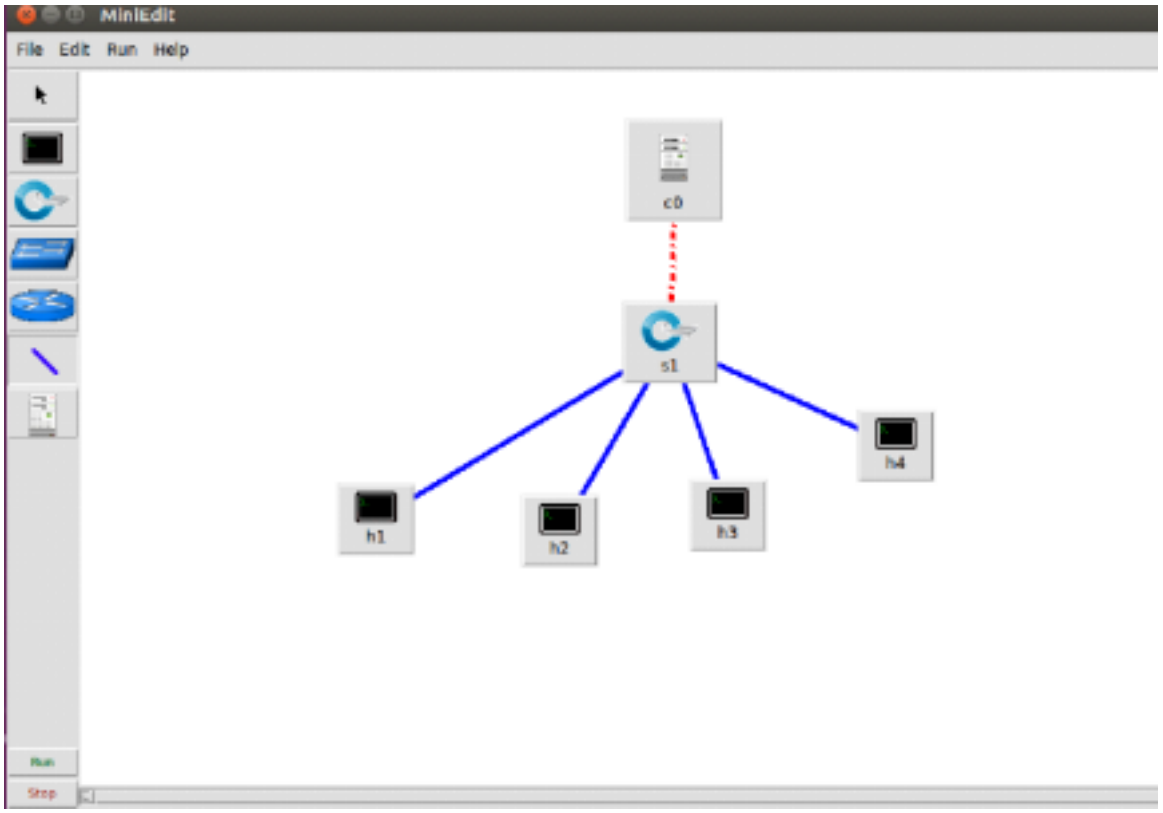

FIG:5.5: NETWORK TOPOLOGY

```
root@admin123-VirtualBox:/home/admin123# wireshark
```

FIG:5.6: STARTING NETWORK SNIFFING USING WIRESHARK AS ROOT USER

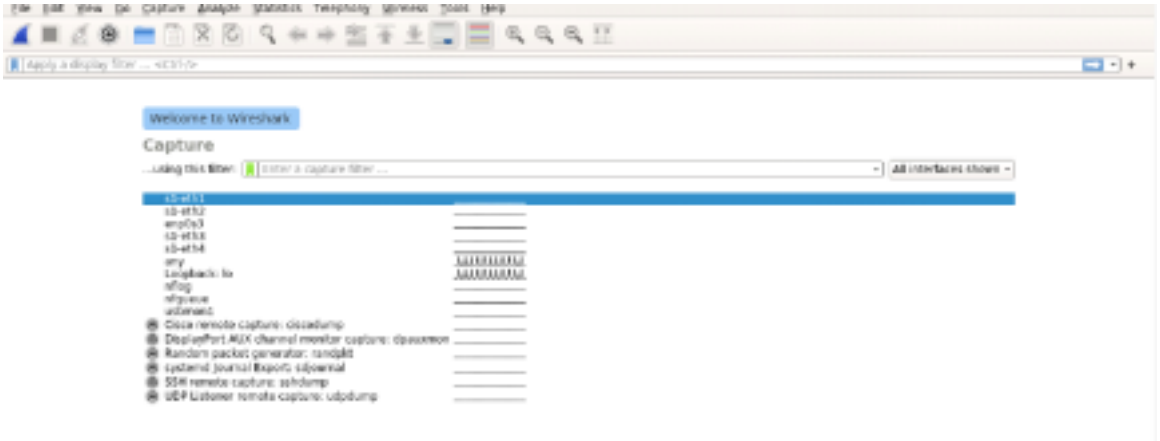

FIG:5.7: INTERFACES IN WIRESHARK

```
mininet> h1 ping h4
PING 10.0.0.4 (10.0.0.4) 56(84) bytes of data.
^C
--- 10.0.0.4 ping statistics ---
92 packets transmitted, 0 received, 100% packet loss, time 93166ms
```

FIG:5.8: Layer - 3 Host-to-Host connectivity blocked, Block all the traffic from Host-h1 to Host-h4

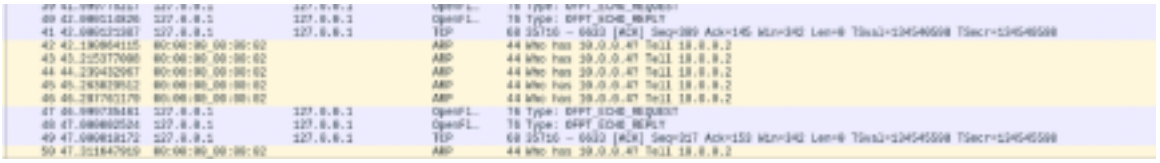

FIG:5.9: IN WIRESHARK THERE IS NO DESTINATION ADDRESS MEANS HOST CONNECTIVITY IS BLOCKED

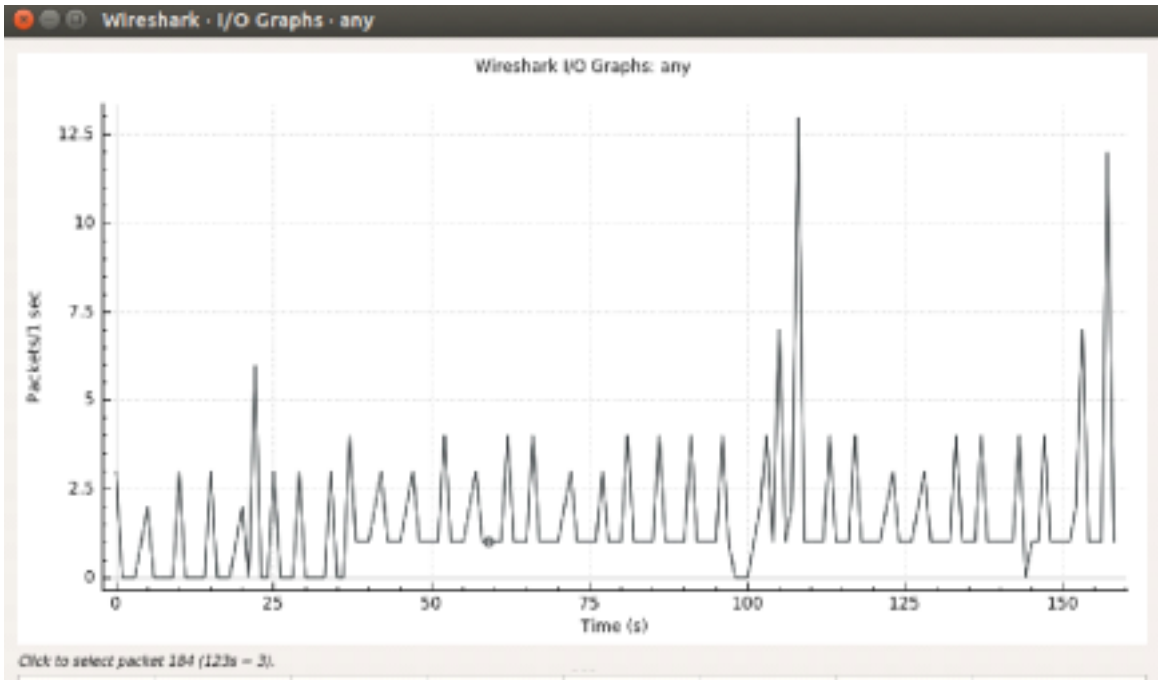

FIG:5.10: STATISTICAL GRAPH FOR LAYER 3 BLOCKAGE OF PACKETS

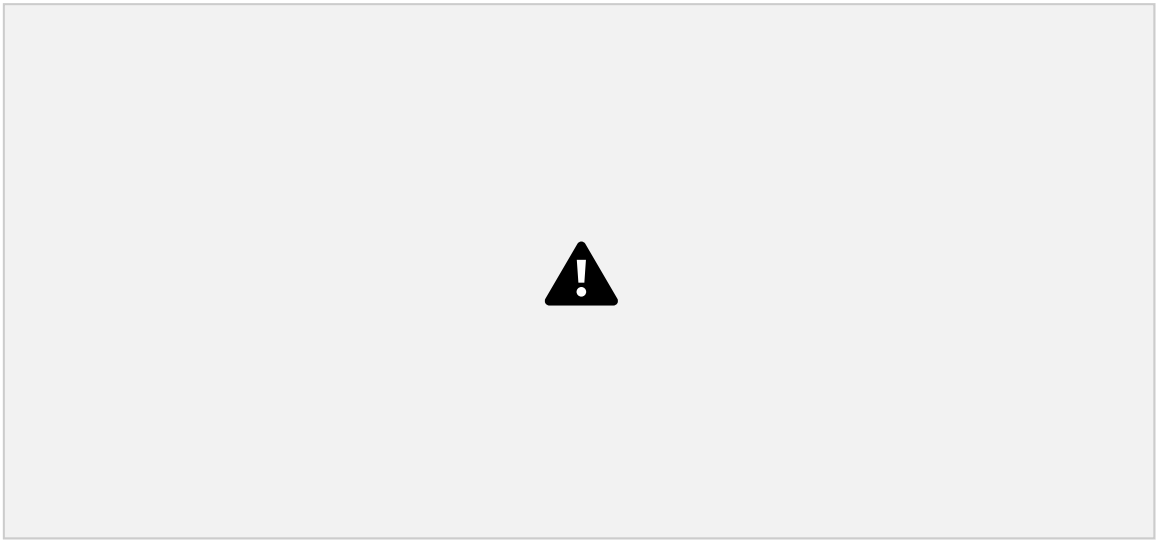

FIG:5.11: Layer - 2 Link connection blocked, Block all the traffic from Host-h2

FIG:5.12: The ARP L2-packets will be dropped by the Controller.

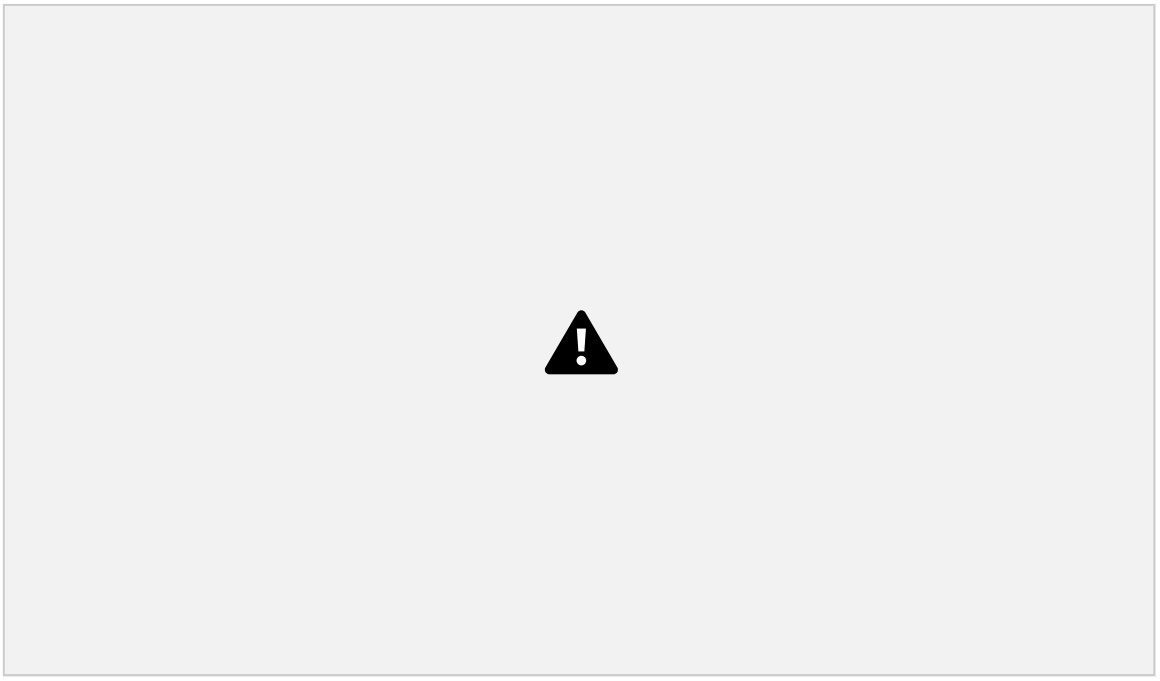

FIG:5.13: STATISTICAL GRAPH FOR LAYER 2 BLOCKAGE OF PACKETS

FIG:5.14: OPENING XTERM TERMINALS FOR HOSTS H1 AND H3

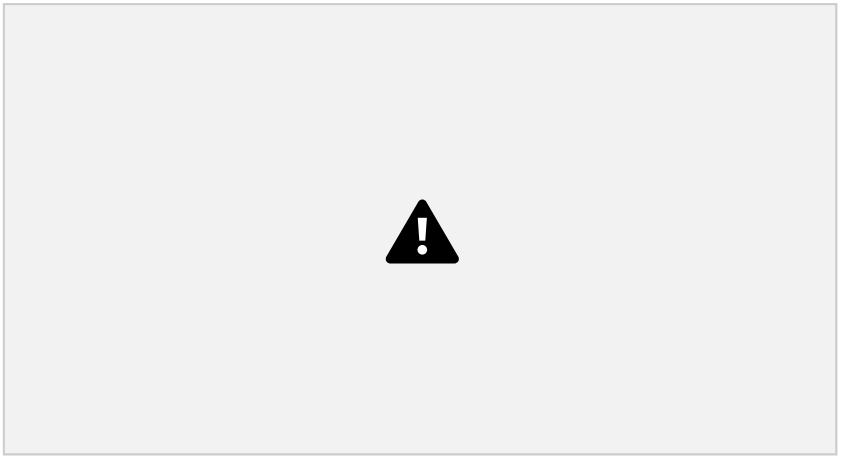

FIG:5.15:Layer - 4 Destination Process blocked, Block all the traffic destined to Process h3 at port 22

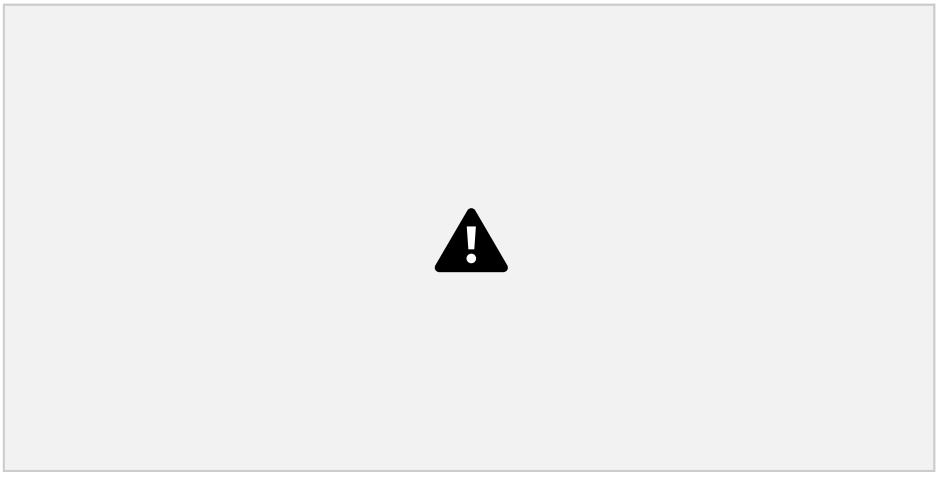

FIG:5.16: NODE 1 REJECTING THE TRAFFIC FROM PORT 22

FIG:5.17:Layer - 4 Destination Process blocked, Block all the traffic destined to Process h3 at port 80

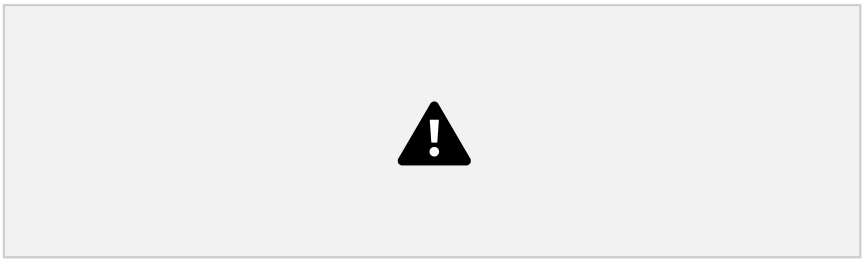

FIG:5.18: NODE 1 REJECTING THE TRAFFIC FROM PORT 80

FIG:5.19: LAYER 4 FIREWALL PACKET INFORMATION

6. CONCLUSION & FUTURE WORK

Firewall is an essential component in networks for security purposes such that it monitors and governs all network traffic and has the ability to identify and block undesired traffic. Deploying physical firewalls might be costly in terms of hardware and updating its system as well. In addition, they lack

flexibility of programmability to administrators whereas software firewalls can do the same functionality and provide programmability.

SDN firewall applications can replace expensive Layer2-7 firewalls, load balance and IDS/IPS. We have successfully implemented a layer2/3/4 SDN firewall running along with layer2_learning switch in the POX controller.

The limitation of our work is that our firewall does not keep record of the state of connection which makes it stateless. The limitation is due to using OpenFlow version 1.0 which does not keep track of the state of the packets as well as only few header fields are supported. Future work can be implementation of stateful firewall. We will also further develop our firewall to be distributed to reduce the overhead on controller in case of much traffic. In future work, some functionalities might be added to this firewall to be capable of performing routing functionality besides firewall functionalities by running it along with modified L3_learning switch.

7.REFERENCES

1.P. Krongbaramee and Y. Somchit, "Implementation of SDN Stateful Firewall on Data Plane using Open vSwitch," 2018 15th International Joint Conference on Computer Science and Software Engineering (JCSSE), 2018, pp. 1-5, doi: 10.1109/JCSSE.2018.8457354.

2.S. Kaur, K. Kaur and V. Gupta, "Implementing openflow based distributed firewall," 2016 International Conference on Information Technology (InCITe) - The Next Generation IT Summit on the Theme - Internet of Things: Connect your Worlds, 2016, pp. 172-175, doi: 10.1109/INCITE.2016.7857611.

3.Network Analysis using Wireshark Cookbook | Packt

https://www.packtpub.com/product/network-analysis-using-wireshark-cookbook/9781849517645

4.W. M. Othman, H. Chen, A. Al-Moalmi and A. N. Hadi, "Implementation and performance analysis of SDN firewall on POX controller," 2017 IEEE 9th International Conference on Communication Software and Networks (ICCSN), 2017, pp. 1461-1466, doi: 10.1109/ICCSN.2017.8230351.